PIPS Is not (just) Polyhedral Software

Mehdi AMINI$^{1,2}$ Corinne AN COUR T$^2$ Fabien COEL HO$^2$
Béatrice CREUS I LLET$^1$ Serge GU E LTON$^{3,2}$ François IRIGO IN$^2$
Pierre JOUVE LOT$^2$ Ronan KERYE LL$^{1,3}$ Pierre VI LLALON$^1$

$^1$HPC Project
$^2$Mines ParisTech/CRI
$^3$Institut TÉLÉCOM/TÉLÉCOM Bretagne/HPCAS

2011/04/03
—
IMPACT 2011
Some archeology

- In the 70’s vector and parallel machines were the only way to get top performances.
- In the 80’s automatic vectorization and parallelization became a hot research topic.
- 1984: Rémi TRIOLET’s PhD @ Mines ParisTech with Paul FEAUTRIER on interprocedural parallelization, convex array regions, polyhedra and linear algebra...
- 1987: François IRIGOIN’s PhD @ Mines ParisTech with Paul FEAUTRIER on tiling, control code generation.
- 1988: PIPS starts as a project to parallelize scientific applications. Motivation: electrocardiography signal processing code written in Fortran.
- 1991: first PIPS PhD: Corinne ANCOURT (on code generation for data communication, under well-known WP65 secret project.)
Some archeology

- Followed a lot of internships, PhDs, post-docs, research engineers...
- Use very French specialties
  - Abstract interpretation to « understand » programs (Cousot, Halbwachs...)
  - Linear algebra to represent things in a mathematical way (good expressiveness, easy to manipulate) (Fourier...)
- Automatic vectorization and parallelization: overly high expectations on alleviated research domains in 90’s–00’s
- Nowadays parallelism here to prevent processors from melting
  - parallel programming is just a way to avoid application to run slower... 😊
- Need parallelism for the masses
- Automatic parallelization is one of the ways to go 😊
- Advanced compilation needed anyway
• PIPS (Interprocedural Parallelizer of Scientific Programs): Open Source project from Mines ParisTech... 23-year old! 😊
• Funded by many people (French DoD, Industry & Research Departments, University, CEA, IFP, Onera, ANR (French NSF), European projects, regional research clusters...)
• One of the projects that introduced polytope model-based compilation
• ≈ 450 KLOC according to David A. Wheeler’s SLOCCount
• ... but modular and sensible approach to pass through the years
  ► ≈300 phases (parsers, analyzers, transformations, optimizers, parallelizers, code generators, pretty-printers...) that can be combined for the right purpose
  ► Polytope lattice (sparse linear algebra) used for semantics analysis, transformations, cone-based dependance graph, code generation... to deal with big programs, not only loop-nests

PIPS Is not (just) Polyhedral Software
Source-to-source to be more independent of targets (trust good work from back-end people 😊)

- NewGen object description language for language-agnostic automatic generation of methods, persistence, object introspection, visitors, accessors, constructors, XML marshaling for interfacing with external tools...
  Cf. presentation @ WIR 2011

- Interprocedural à la `make` engine to chain the phases as needed.
  Lazy construction of resources

- On-going efforts to extend the semantics analysis for C

- Around 15 programmers currently developing in PIPS (Mines ParisTech, HPC Project, IT SudParis, TÉLÉCOM Bretagne) with public `svn`, Trac, `git`, mailing lists, IRC, Plone, Skype... and use it for many projects
Current PIPS usage

- Automatic parallelization (Par4All C & Fortran to OpenMP)
- Distributed memory computing with OpenMP-to-MPI translation [STEP project]
- Generic vectorization for SIMD instructions (SSE, VMX, NEON, CUDA, OpenCL...) (SAC project) [SCALOPES, SMECY]
- Parallelization for embedded systems [SCALOPES, SMECY]
- Compilation for hardware accelerators (Ter@PIX, SPoC, SIMD, FPGA, SCMP, MPPA...) [FREIA, SCALOPES, SIMILAN]
- High-level hardware accelerators synthesis generation for FPGA [PHRASE, CoMap]
- Reverse engineering & decompiler (reconstruction from binary to C)
- Genetic algorithm-based optimization [Luxembourg university+TB]
- Code instrumentation for performance measures
- GPU with CUDA & OpenCL [TransMedi@, FREIA, OpenGPU MediaGPU, SMECY]
Outline

1. Key use cases
2. Key PIPS internals
3. Code transformations for heterogeneous computing
4. Conclusion
Vectorization and parallelization

- Historical application for PIPS (1988–)
  - Introduced interprocedural parallelization based on linear algebra method
  - Fortran 77 → Cray Fortran, CM Fortran, Fortran 90 array syntax, HPF, OpenMP loops
  - Fine grain, coarse grain, loop nest...

- Come back with SIMD instruction sets in most recent processors
  - SAC (SIMD Architecture Compiler) in PIPS (2003–2011)
  - Based on unrolling and SLP extraction instead of direct vectorization
  - Generate source with vector types & intrinsic functions for x86 SSE/AVX, ARM NEON (smart phones, tablets)...
  - Useful in GPU too: generate OpenCL & CUDA vector data types and intrinsics

Cf. Adrien GUINET’s poster @ CGO 2011
Key use cases

- Transputer-based parallel computer
  - Automatic code parallelization
  - Distribution of sequential code
  - "Compile" a global shared memory with some nodes running computations and some other giving memory services
  - Introduced
    - Code generation by scanning polyhedra
    - Code distribution with a linear algebra method
  - PVM version too
- More recently, generation of SPMD MPI code from OpenMP code by using PIPS convex array regions [STEP @ Institut Télécom SudParis]
HPF compilation

- Extension of WP65 concepts to HPF compilation (1992–1997)
- HPF = Fortran + Arrays of processors + Affine data-mapping of arrays

```fortran
real A(0:24), B(0:24) ! 0 \leq a_A \leq 24, 0 \leq a_B \leq 24

!HPF$ template T(0:80) ! 0 \leq t \leq 80

!HPF$ processors P(0:3) ! 0 \leq p \leq 3

!HPF$ align A(i) with T(3*i) ! a_A = 3t

!HPF$ align B(i) with A(i) ! a_A = a_B

!HPF$ distribute T(cyclic(4)) onto P ! t = 16c + 4p + \ell

! 0 \leq \ell < 4

A(0:U:3) = A(0:U:3) + B(1:U+1:3) ! i = 3i/\!, 0 \leq i \leq U

! a = i
```
Key use cases

- **HPF compilation**
  - Distribute code and data on processors without shared memory
  - Generate allocations, local iterations, optimize communications, remappings and IO

PIPS Is not (just) Polyhedral Software

IMPACT 2011 — 2011/04/03

Ronan KERYELL et al.
HPF compilation

- Array distribution:

\[
\text{own}_X(p) = \{ a | \exists t, \exists c, \exists \ell : \begin{align*}
R_X t &= A_X a + t_0 \\
\Pi t &= C_X P_c + C_X p + \ell_X \land 0 \leq a < D_X \\
0 \leq p < P &\land 0 \leq \ell < C_X \\
0 \leq t < T_X \}
\]

- Local iterations (owner compute rule):

\[
\text{compute}(p) = \{ i | S_X i + a_{X0} \in \text{own}_X(p) \}
\]

- Elements needed by computation:

\[
\text{view}_Y(p) = \{ a | \exists i \in \text{compute}(p) : a = S_Y i + a_{Y0} \}
\]
Key use cases

HPF compilation

- Send-receive
  \[send_Y(p) = \{(p', a) \mid a \in own_Y(p) \cap view_Y(p')\}\]
  \[receive_Y(p) = \{(p', a) \mid a \in view_Y(p) \cap own_Y(p')\}\]

- Compact allocation (HERMITE + non-linear transformation)

- Extension to Phénix machine from ETCA/SEH (work with Pierre Fiorini, CEO of HPC Project)

- Coming back? Placement directives interesting nowadays to organize manycore data and computations...
Key use cases

- Providing high level tools: direct compilation of sequential code
- Adaptation of previous techniques
  - Generate host and accelerator code from pragma annotated code (CoMap) (2004–2007)
  - Generalize and improve for Ter@pix vector accelerator from THALES (2008–2011)
  - Support of CEA SCMP task oriented data-flow machine (2011)
  - Par4All project for GPU and other manycore accelerators (ST Microelectronics P2012, Kalray MPPA...) (2010–)
- Configurations for the SPoC configurable image pipelined processor
  Cf. Fabien Coelho’s presentation @ ODES 2011
Key use cases

Program Verification

- Automatic parallelization and abstract interpretation in PIPS: uses verifiers of mathematical polyhedral proofs
- Can also be used
  - To extract semantics properties to prove facts about programs
  - Array bound checking and provably redundant array bound checks removing
  - On-going more precise linear integer pre- and post-conditions on programs

Cf. François IRIGOIN presentation @ ACCA 2011
Program synthesis

- Code generation and memory allocation from application descriptions in SPEAR-DE from THALES
- Composition of Simulink, Scade, Xcos/Scicos components by analyzing the C code of components (HPC Project 2010–)
Key use cases

High-level hardware synthesis

- Generate FPGA configurations from sequential code + pragma (2002–2004)
- Use Madeo hardware synthesis tool from UBO, SmallTalk as input language
- Side effect: SmallTalk prettyprinter in PIPS 🙃
Key use cases

- Decompilation
  - Parallelization of binaries?
  - Generate raw C-equivalent code with `objdump` + HPC Project crude C translator (2008)
  - Apply PIPS code restructurer (control graph restructuring, graph loop recovering...)
  - Apply PIPS parallelization
Outline

1. Key use cases
2. Key PIPS internals
3. Code transformations for heterogeneous computing
4. Conclusion
General organization

- **Compiler & tools:** p4a (Par4All), sac (SIMD), terapyps (Ter@pix)
- **Pass manager:** PyPS, tpips
- **PIPSmake consistency manager**
- **Phases**
  - Passes: inlining, unrolling, communication generation...
  - Analyses: HCFG, DFG, array regions, transformers, preconditions...
  - Prettyprinters: C, Fortran, XML...
- **Internal representation**
  Cf. Fabien Coelho’s presentation @ WIR 2011
Simple memory effects

- Describe memory operations performed by a given statement
- **Proper effects**: memory references local to individual statements
- **Cumulated effects** take into account all effects of compound statements, including those of their sub-statements
- **Summary effects** summarize the cumulated effects for a function and mask effects on local entities

```c
int corr(int N, float x[N], float y[N], int M, float R[M])
{
    if (M < N) {
        // <may be read>: N k x[*] y[*]
        // <may be written>: R[*]
        // <is read>: M
    // <is written>: k
```
Simple memory effects

```c
for (int k = 0; k <= M-1; k += 1)
    // <may be read>: x/*] y/*
    // <may be written>: R[*]
    // <is read>: MN k
    R[k] = corr_body(k,N,&x[k],y);
}
return 1;
```

```c
} else
    return 0;
}
```
Transformers

- Basis for *linear relation analysis* in PIPS
- Represent relation between the store after an instruction and the store before in a linear way (mainly for integer variables)

```c
// T() {}
float corr_body(int k, int N, float x[N], float y[N]){
  // T() {}
  float out = 0.;
  // T(n) {k+n'==N}
  int n = N-k;
  // T(n) {k+n==N,1<=n ', n'<=n,1<=n}
  while (n>0) {
    // T(n) {n'==n-1,k+1<=N,0<=n '}
    n = n-1;
    // T() {k+1<=N,0<=n}
    out += x[n]*y[n]/N;
  }
  // T() {k+n<=N, n<=0}
  return out;
}
```
Can be used by `forloop_recover` transformation:

```c
float corr_body(int k, int N, float x[N], float y[N]){
    float out = 0.;
    int n = N-k;

    for(int n0 = n; n0 >= 1; n0 += -1) {
        n = n0 -1;
        out += x[n]*y[n]/N;
    }
    return out;
}
```
Preconditions

- Affine predicates over scalar variables
- Computed by combination of transformers
- Interprocedural analysis
- Used in many phases (partial evaluation, dead code elimination...)

```c
// P() {k+2<=N, 0<=k}
float corr_body(int k, int N, float x[N], float y[N]) {
    // P() {k+2<=N, 0<=k}
    float out = 0.;
    // P() {k+2<=N, 0<=k}
    int n = N-k;
    // P(n) {k+n==N, k+2<=N, 0<=k}
    while (n > 0) {
        // P(n) {k+2<=N, k+n==N, 0<=k, 1<=n}
        n = n - 1;
        // P(n) {k+2<=N, k+n+1<=N, 0<=k, 0<=n}
        out += x[n]*y[n]/N;
    }
    // P(n) {n==0, k+2<=N, 0<=k}
}```
Preconditions

```cpp
return out;
```

...
Convex array regions

- Abstract with affine equalities and inequalities set of array elements accessed by statement
- Many different model of regions: read/write/in (needed)/out (useful after)/...

```c
int corr(int N, float x[N], float y[N],
    int M, float R[M]) {
    // <R[PHI1]−W-MAY−{0<=PHI1, PHI1+1<=M,M+1<=N}>
    // <x[PHI1]−R-MAY−{0<=PHI1, PHI1+1<=N,1<=M,M+1<=N}>
    // <y[PHI1]−R-MAY−{0<=PHI1, PHI1+1<=N,1<=M,M+1<=N}>
    if (M<N) {
        // <R[PHI1]−W-EXACT−{PHI1==k,0<=k , k+1<=M,M+1<=N}>
        // <x[PHI1]−R-EXACT−{k<=PHI1, PHI1+1<=N,0<=k , k+1<=M,M+1<=N}>
        // <y[PHI1]−R-EXACT−{0<=PHI1, PHI1+k+1<=N,0<=k , k+1<=M,M+1<=N}>
    }
    // ...
```
Convex array regions

```
kernel(M,N,k,R,x,y);
}
return 1;
}
else
return 0;
}
```
Key PIPS internals

Linear algebra for analyses and transformations

- PIPS analyses based on the $C^3$ linear algebra library
- Mainly developed at MINES ParisTech from the 80's
- Integer vectors, matrix, polynomial...
- Mathematical operations, HERMITE's normal form, SMITH's normal form, sorting, simplex...
- Implementation of all the PIPS polyhedral and linear analyses and transformations (unimodular transformations...)
- In real code, large number of variables including global variables that are mostly not related
  - Use a sparse representation of constraints: reduce memory storage
Consistency and persistence manager

- Many passes and resources in PIPS...
- Difficult to have always up-to-date informations
- Consistency manager using an à la make description of dependence relations between resources though passes or analyses
- Lazy construction of resources to produce goal asked by user
- Deal with interprocedural analysis
- A persistence manager allows to stop and resume PIPS later
Pass manager

- PIPS is a source-to-source tool box
- ...but how to use them?
- Simple `tpips` shell like
- New Python-based PyPS
  - Modules, loops and compilation units are exposed as first-class entities
  - Introspection
  - Base of Par4All
Cf. PIPS tutorial @ CGO 2011
Outline

1. Key use cases
2. Key PIPS internals
3. Code transformations for heterogeneous computing
4. Conclusion
Computation intensity estimation

- Offloading a loop on accelerator or not?
- Relevant only if the data transfer vs. computational intensity trade-off is interesting
- Execution time estimation given by complexity analysis
- Memory size estimated by region analysis as a polynomial in the program variables
Outlining

- Off-loading to accelerator...
- Use load work store idiom
- Extract work into new functions to be executed on accelerator
- Use summary effects to build formal parameters
- Use privatization analysis to filter out variables with local use only
Statement Isolation

- Isolate all data accessed by a statement in newly allocated memory areas: simulate the remote memory
- Use *convex array regions* to generate the data copy between the remote and local memories
- DMA can often only transfer efficiently rectangular areas: over-estimate regions using their rectangular hull
- *read regions* are translated into a sequence of host-to-accelerator data transfers
- *written regions* are converted into accelerator-to-host data transfers

Cf. PIPS tutorial @ CGO 2011
Rectangular symbolic tiling and memory footprint

- Array regions estimate memory needed for a computation
- If it exceeds accelerator memory size, cannot run in 1 pass
- Use some tiling, but depends on memory needed
- Perform symbolic tiling
- Compute memory footprint according to tiling parameters
- If not possible to decide at compile time, postpone at run time
Parallel loop nests are compiled into a CUDA kernel wrapper launch.

The kernel wrapper itself gets its virtual processor index with some `blockIdx.x*blockDim.x + threadIdx.x`.

Since only full blocks of threads are executed, if the number of iterations in a given dimension is not a multiple of the `blockDim`, there are incomplete blocks.

An incomplete block means that some index overrun occurs if all the threads of the block are executed.
So we need to generate code such as

```c
void p4a_kernel_wrapper_0(int k, int l,...)
{
    k = blockIdx.x*blockDim.x + threadIdx.x;
    l = blockIdx.y*blockDim.y + threadIdx.y;
    if (k >= 0 && k <= M - 1 && l >= 0 && l <= M - 1)
        kernel(k, l, ...);
}
```

Guard ≡ directly translation in C of preconditions on loop indices that are GPU thread indices

```c
// P(i, j, k, l) {0<=k, k<=63, 0<=l, l<=63}
```
Outline

1. Key use cases
2. Key PIPS internals
3. Code transformations for heterogeneous computing
4. Conclusion
Manycores & GPU: impressive peak performances and memory bandwidth, power efficient

Future will be heterogeneous

Programming tools will be heterogeneous too: association of different tools specialized in different domains

Future challenge: composing tools to make robust compilers

PIPS uses polyhedral abstractions at high-level with approximations
  
  Prefer to deal with whole programs rather than optimal method on small parts (work done in a Mining school, not École Normale Supérieure 😊)
  
  Good to prepare work for other more specialized and precise tools
  
  On-going interfacing with PoCC in OpenGPU project

Source-to-source
  
  Avoid sticking to much or architectures
Conclusion

- But can also capture architectural details
- Source is a great way to interface ≠ tools!
- Extensions in Python with more abstractions and dynamicity
- Basis of Par4All tool to provide end-user tools
- Open Source for community network effect
- More information this afternoon on PIPS and Par4All during the tutorial
Questions?

Historical disclaimer
I’m related to this project for only 19 years, so I ignore many details from the beginning but some colleagues in the audience can answer 😊

Completeness disclaimer
- There are too many things in PIPS and nobody knows about all of them anyway 😊
- Not enough things has been published on PIPS 😞